VulnerableMe: Measuring Systemic Weaknesses in Mobile Browser Security

Chaitrali Amrutkar[1], Kapil Singh[2], Arunabh Verma[1], and Patrick Traynor[1]

[1] Georgia Tech Information Security Center (GTISC), Georgia Institute of Technology
(chaitrali@, arunabh.verma@ traynor@cc.)gatech.edu
[2] IBM Research
kapil@us.ibm.com

Abstract. Porting browsers to mobile platforms may lead to new vulnerabilities whose solutions require careful balancing between usability and security and might not always be equivalent to those in desktop browsers. In this paper, we perform the first large-scale security comparison between mobile and desktop browsers. We focus our efforts on display security given the inherent screen limitations of mobile phones. We evaluate display elements in ten mobile, three tablet and five desktop browsers. We identify *two new classes of vulnerabilities* specific to mobile browsers and demonstrate their risk by launching real-world attacks including display ballooning, login CSRF and clickjacking. Additionally, we implement a new phishing attack that exploits a default policy in mobile browsers. These previously unknown vulnerabilities have been confirmed by browser vendors. Our observations, inputs from browser vendors and the pervasive nature of the discovered vulnerabilities illustrate that new implementation errors leading to serious attacks are introduced when browser software is ported from the desktop to mobile environment. We conclude that usability considerations are crucial while designing mobile solutions and display security in mobile browsers is not comparable to that in desktop browsers.

1 Introduction

Mobile web browsers have long underperformed their desktop counterparts. Whether by implementing limited alternative standards such as WAP [44] or incomplete versions of HTML, the first mobile browsers provided a meager set of capabilities and attracted only a small number of early adopters. However, recent improvements in processing power and bandwidth have spurred significant changes in the ways users experience the mobile web.

Modern mobile browsers now build on the same or similarly capable rendering engines used by many desktop browsers [12, 13]. Mobile browsers are so capable that, through APIs such as WebViews, many of the most popular mobile apps (e.g., Facebook, ESPN) act as wrappers for the browser pointed to specific webpages. However, due to limitations in the screen real estate and memory, existing desktop browser software was not directly ported to mobile devices. Accordingly, while many mobile browsers bear the name of related desktop applications, their internal components are significantly different. The impact of these changes on security has not previously been evaluated. Given the popularity of browsing on mobile devices [26,36], focusing on the security of mobile browsers is critical.

In this paper, we perform the first large-scale security comparison between mobile and desktop browsers. While there are many potential areas for investigation, we focus on the issues of display security due to the screen constraints of mobile devices. Given the often crowded layout of mobile webpages, we specifically investigate the behavior of overlapping HTML elements (and how browsers handle clicks - i.e., "user event routing"), behavior at the boundaries between non-overlapping items ("boundary control") and the impact of nonpersistent availability or complete absence of the address bar. We apply blackbox analysis across ten mobile, three tablet and five desktop browsers and demonstrate that many mobile and tablet browsers are vulnerable to new two classes of attacks due to inconsistent click-event routing and incorrect write policies. We illustrate that desktop browsers are not susceptible to these attacks and present solutions to address the new vulnerabilities. We then discover a third class of vulnerability resulting from a clash between considerations made for usability in mobile browsers and a universally implemented display policy, demonstrating that making usability considerations while creating mobile software is crucial and blind porting of traditional browser code to mobile devices can introduce unexpected vulnerabilities.

We make the following contributions:

- **Characterize display security disparity between the most popular mobile and desktop browsers:** We analyze display security on ten mobile (Android Mobile, Blackberry (Mango), Blackberry (Webkit), Chrome Beta, Firefox Mobile, Internet Explorer (IE) Mobile, Nokia Mini-Map, Opera Mini, Opera Mobile and iPhone Safari), three tablet (Android on Motorola Xoom, Android on Samsung Galaxy and iPad2 Safari) and five desktop (Chrome, Firefox, Internet Explorer, Opera and Safari) browsers. We use blackbox analysis as source code is not available for the majority of browsers. Table 2 on page 14 summarizes our findings.
- **Identify erroneous implementations of display security policies:** We identify *previously unknown* erroneous policies in user event routing and boundary control and implement multiple attacks that demonstrate their seriousness. Even though many mobile browsers rely on the same rendering engines as their desktop counterparts, our experiments demonstrate that mobile browsers are vulnerable to attacks not previously seen in the desktop space.
- **Expose conflict between usability and display security:** We show that some re-implemented policies from desktop browsers, specifically Top-Level Frame Navigation [21], expose mobile devices to phishing when mobile browsers hide or completely eliminate indicators such as the address bar for reasons of usability. In particular, we demonstrate the ability to navigate users away from their intended destinations. *Our technique is new and does not use address bar spoofing similar to the phishing techniques studied earlier [30, 37].* We find that our technique enables a more dangerous and easy to launch attack, since it exploits a built-in policy in all web browsers instead of attempting to spoof the address bar in individual browsers.

Our analysis demonstrates that the discovered vulnerabilities are not isolated bugs; rather, *they are pervasive and affect all but one of the most popular mobile and tablet browsers in some capacity.* We have communicated our results to various browser vendors who have acknowledged the presence of these vulnerabilities. Moreover, we argue that because an increasing number of apps rely on mobile browsers, that these issues are

Category	Browser Name	Version	Rendering Engine	Operating System	Device
Mobile	Android	2.3.6	Webkit	Android 2.3.6	Nexus One
	Blackberry	5.0.0	Mango	Blackberry OS 5.0.0.732	Bold 9650
	Blackberry	6.0.0	Webkit	Blackberry OS 6	Torch 9800
	Chrome Beta	0.16.4301.233	Webkit	Android 4.0	Galaxy Nexus
	Firefox Mobile	4 Beta 3	Gecko	Android 2.3.6	Nexus One
	Internet Explorer Mobile	*	Trident	Windows Phone 7.0.7004.0 OS	LG-C900
	Nokia Mini-Map	*	Webkit	Symbian S60	E71x
	Opera Mini	6.0.24556	Presto	Android 2.3.6	Nexus One
		5.0.019802	Presto	iOS 4.1 (8B117)	iPhone
	Opera Mobile	11.00	Presto	Android 2.3.6	Nexus One
	Safari	*	Webkit	iOS 4.1 (8B117)	iPhone
Tablet	Android	*	Webkit	Android 3.2.1	Motorola Xoom
	Android	*	Webkit	Android 3.1	Samsung Galaxy
	Safari	*	Webkit	iOS 4.3.5 (8L1)	iPad 2
Desktop	Chrome	15.0.874.106	Webkit	OS X 10.6.8	–
	Firefox	7.0.1	Gecko	OS X 10.6.8	–
	Internet Explorer	8.0.7600.16385	Trident	Windows 7	–
	Opera	11.52	Presto	OS X 10.6.8	–
	Safari	5.1.1	Webkit	OS X 10.6.8	–

Table 1. Details of the browsers used for experimental evaluation. We also evaluated Opera Mini 5.5.1, Android 2.2.1 and Android 2.3.3 on Nexus One and Android 4.0.1 on Galaxy Nexus. We observed the same vulnerabilities in both the old and new versions of Opera Mini and Android browsers (except Android 4.0.1). Android 4.0.1 is susceptible to attacks in Section 3 and Section 5, but not from Section 4. (*: The version numbers of these browsers were not apparent. We have used the default browsers shipped with the referenced version of the OS.)

relevant to all mobile app developers. Our results are the first *comprehensive* study in display security and they provide strong evidence that the security of mobile browsers has taken steps backward when compared to desktop browsers.

2 Overview

This section discusses our experimental methodology and defines our threat model.

2.1 Methodology

We analyze the rendering differences between popular desktop and mobile browsers for security. The studied browsers are shown in Table 1. We have selected these browsers as they represent approximately 90% of mobile browsers in the market [7].

We define a 'display element' as any HTML element that can color pixels on the screen. For example, `iframe`, `image`, `text`, `text area`, `link`, `table` and `button` all fall under display elements. However, HTML elements such as `head` or `option` do not qualify as display elements. We create customized scenarios to evaluate common interactions of cross-origin display elements: 1) when they overlap, 2) when they border each other and 3) when they are navigated to new sources. Given the tight layout of many mobile webpages and the corresponding small screen sizes of the associated devices, characterizing such interactions is critical. We discover new classes of vulnerabilities in mobile browsers and evaluate their risk by implementing attacks exploiting the vulnerabilities. All the experiments were performed on browsers on real mobile phones, and are recreated in the respective emulators to create many of the figures throughout the paper.

2.2 Threat Model

We consider two classes of adversaries. Each adversary attempts to attack other website principals and/or the user and exploit the constrained nature of a mobile de-

vice's display. Each adversary can identify the user's mobile browser and is knowledgeable of the display-related security vulnerabilities associated with that browser.

Landlord attacker: The *landlord attacker* is a malicious principal[3] who can host his own websites such as `landlordattacker.com`. For example, the owner of a phishing website such as `blankofamerica.com` imitating `bankofamerica.com` is classified as a landlord attacker. A 'tenant' is a principal who rents an area on a landlord's website to render his own content such as advertisements. After the landlord gets honest tenants on his website, he attempts to exploit the honest tenant and/or the honest user. The landlord cannot read or change parts of the content in the tenant's rented area on the screen (due to the Same Origin Policy[4]), but controls the external properties of the tenant's rented area. For example, the landlord can specify the dimensions, transparency and position of the tenant's area on his website. The landlord instead tries to attack the honest tenant and honest user by manipulating his own website display.

We note that not every user visiting the malicious website will be exploited. Depending on the vulnerability targeted by the landlord attacker, the honest tenant and honest user may be attacked only when `landlordattacker.com` is rendered in a vulnerable browser. Placing web advertisements, displaying popular content indexed by search engines and sending bulk e-mail to users are some of the techniques that the landlord attacker can use to attract users to his website [23].

Tenant attacker: The *tenant attacker* is a malicious principal who can rent an area of the display on a website owned by an honest landlord. For example, the tenant attacker can insert a malicious advertisement or widget into an honest website. Websites such as iGoogle allow any user having an account to upload a new widget. We assume that an honest user visits an honest website containing at least one tenant attacker area using a vulnerable mobile browser. The tenant attacker has knowledge of the display vulnerabilities in the popular mobile browsers. He manipulates the content of his rented area to attack the honest website and/or the user.

A successful exploit is able to (1) influence the state and logic of a victim website principal across Same Origin Policy boundaries, and/or (2) deceive a user into performing unintended actions or sharing private data.

3 User Event Routing

Overlapping elements are common in many webpages. From drop-down menus to floating advertisements, the ability to overlay objects allows for content to be dynamically presented to the user. However, the interaction between such elements must be strictly defined, especially in cases when they are controlled by different origins. When two or more display elements share the same pixel on the screen, browsers must decide both a) which element can control the 'coloring' (display) of the pixel and b) which element owns and responds to the user access to that pixel (user event routing). For example, if a drop-down menu covers over an image and a user clicks in this shared screen

[3] A principal is the owner of some web content. In general, one principal does not trust another with respect to his resources [46].

[4] The Same Origin Policy prevents a document or script loaded from one domain from getting or setting properties of a document from another domain [10, 38].

area, the browser must decide whether the principal owning the image or the principal owning the menu will respond to a user's click action.

Although all browsers make these decisions, the security relevance of user event routing in overlapped elements has not previously been studied. Our evaluation demonstrates that while desktop browsers consistently route user actions to the topmost element, event routing is inconsistent across mobile and tablet browsers. *This inconsistency allows hidden elements to intercept user actions and potentially perform dangerous operations.* We first discuss the results of our evaluation of overlapped elements using the methodology in Section 2.1 and then present attacks exploiting the vulnerabilities.

3.1 Experimental Evaluation

Mobile and tablet browsers:
Inconsistent click-event reception: Click-event reception refers to a browser choosing the element that receives a user's click action in a stack of overlapped elements. In the Android mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers, a user's `onclick` event on an image is routed to the `onclick` events of buttons, text areas and links below the opaque image, thereby executing the events of the hidden elements. We note that only the events corresponding to the element directly situated below the area where a user clicks responds to the click action. Click events of all the elements situated below the image are not executed when the user clicks on the image.

In the Nokia Mini-Map and Opera Mini browsers, even if the top image has an `onclick` event associated with it, the `onclick` events of the buttons below the image are given preference. If the image on top does not have an event associated with it, the buttons below the image are clickable in the Android mobile and Android tablet on Xoom browsers.
Incorrect write policy: The Android mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers allow a user to write into the text areas in an iframe situated below an opaque image. When a user clicks on the portion of the image overlapping any part of the text area below, the text area pops out on top and the user can write into the box.

Desktop browsers: The desktop browsers always route click and write events exclusively to the top element in a stack of overlapped elements.

3.2 Attacks

We present three novel techniques that exploit inconsistent click-event reception and incorrect write policies for overlapping elements.
1) Click Fraud: This attack is possible due to inconsistent click-event reception in overlapping elements. Click fraud occurs in pay-per-click advertising when a malicious principal creates illicit clicks on an ad by either tricking a real user or by imitating a legitimate user's click with a program. Such attacks generate revenue per click with no actual interest in the target of the ad's link. A popular pay-per-click advertising program is Google's AdSense. A malicious landlord or tenant website cannot manipulate the ad placed by Google (due to the Same Origin Policy) and thus cannot trick a legitimate user into clicking on an unwanted ad by disguising it with more enticing content.

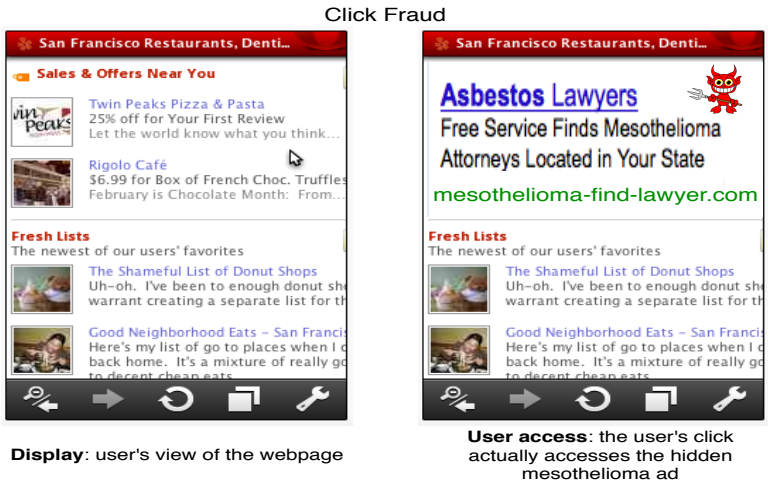

Fig. 1. **Left image**: Fake image advertisement of sales in San Francisco on the `www.landlordattacker.com` website; **Right image**: The mesothelioma ad from Google AdSense placed directly below the enticing fake sales ad image by malicious landlord. A user clicking on the mesothelioma ads [1] earns the landlord attacker more money. The landlord places the honest mesothelioma ads from AdSense in an iframe and overlays it with the more enticing images of sales in San Francisco to increase the rate of clicks. When a user clicks on the fake sale ad in San Francisco, the mesothelioma ad is clicked benefiting the landlord attacker. The Opera Mini (pictured), Android mobile, Android tablet on Xoom and Nokia Mini-Map browsers are vulnerable to the click fraud attack.

Consider a malicious landlord principal who creates an AdSense account and embeds relevant content containing targeted keywords to attract high paying ads. The high paying ads [1] are generally not as popular as ads for discounts or coupons and thus are not clicked very often. A landlord attacker can carry out click fraud as shown in Figure 1, on a browser that allows a user to inadvertently access hidden content (links, buttons etc.) placed below an opaque element such as an image. The landlord attacker overlaps the mesothelioma ad (right) with more enticing and opaque content such as sales at local restaurants (left). If an honest user clicks the area containing the attractive content from a vulnerable browser, the mesothelioma ad[5] below the attractive content will be clicked without the user's knowledge. Since the user's click is captured by the Google AdSense ad instead of the image on top, the malicious landlord illicitly benefits.

2) Login CSRF: This attack is possible due to inconsistent click-event reception and incorrect write policies. The intention of an attacker in a login Cross Site Request Forgery (CSRF) is to make the honest user's browser log in *as the attacker* into a legitimate website without any notice to the user. While seemingly counter-intuitive, such an attack allows an adversary to monitor operations executed by the user and steal their private information. For example, if an attacker successfully logs in into his Yahoo account from the victim's browser, the victim's actions on all of the websites (search, shopping, finance, health) belonging to Yahoo's single sign-on system will be recorded in the attacker's account. If the user makes a purchase at `shopping.yahoo.com` and enters his credit card details, the information will be stored in the attacker's profile. Note that

[5] Mesothelioma is a cancer caused by inhaling asbestos and an ad costs $65.21 per click [9].

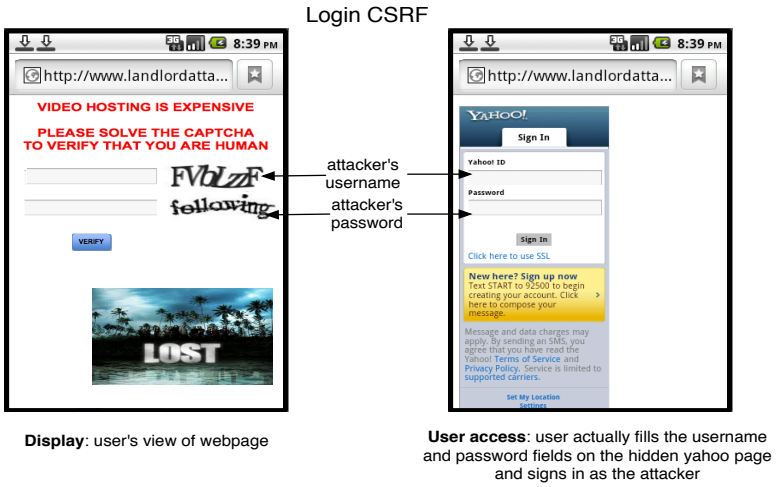

Fig. 2. **Left image**: Image overlapping the `www.yahoo.com` iframe on `www.landlordattacker.com`. The text areas for entering 'solution' of the CAPTCHAs are placed exactly over the email and password fields on `yahoo.com`. The verify button is placed exactly above the 'sign in' button of `yahoo.com`. The two CAPTCHAs are the real email and password of the attacker's Yahoo account.; **Right image**: Login page of `www.yahoo.com` included in an iframe on `www.landlordattacker.com`, placed below the image. The Android mobile (pictured), Android tablet on Xoom, Opera Mini and Nokia Mini-Map browsers are vulnerable to this attack.

the user will not be asked to sign-in since the attacker has already signed in in the user's browser. Previous work has leveraged a browser's network connectivity and a browser's state to launch a login CSRF attack [22]. We present a new mechanism to launch the login CSRF attack by exploiting the vulnerability of incorrectly handling user access to overlapped display elements in mobile browsers. Our method is more robust and not easy to detect since it exploits an in-built vulnerability in the browsers.

Consider a malicious website `landlordattacker.com`. The landlord includes a legitimate iframe containing the 'sign in' page of `www.yahoo.com` as shown in Figure 2 (right). The landlord then overlaps the iframe completely with an opaque image as shown in Figure 2 (left). The image shows enticing free content on the landlord's website and includes two image CAPTCHAs expected to be solved by the user to access the free content. The intention of the landlord attacker is to make the user enter the attacker's credentials into the hidden iframe below the opaque image. The landlord accomplishes this by setting the two CAPTCHAs to the email and password of the attacker's Yahoo account. For example, in Figure 2, *FVbLzzF* and *following* are the username and password respectively of the attacker's Yahoo account. The landlord attacker then carefully places each of the solution boxes of the CAPTCHAs on the image exactly overlapping the email and password fields (text areas) of the Yahoo iframe below the opaque image. The 'Verify' button on the image of the CAPTCHAs is exactly overlapped with the 'Sign in' button of the Yahoo iframe below.

When an honest user visits `landlordattacker.com` from a vulnerable browser, he solves the two CAPTCHAs on the image to view free content. Since the browser allows user access to the text area below the image, when the user fills in the CAPTCHA on top, *he actually fills in the username and password of the landlord attacker in the*

Yahoo iframe below the image. Once the user clicks the verify button on the image, the 'sign in' button on the Yahoo iframe is clicked instead, thereby logging the user's browser into `www.yahoo.com` as the attacker.

In general, solving a CAPTCHA does not disclose private user information and is perceived as a security feature. Therefore, even a careful user would likely be willing to solve the CAPTCHA. Because the top image is opaque, the user is completely oblivious to the consequences of his seemingly benign action. Once the attacker is logged in from the user's browser, all the potential consequences of login CSRF are possible.

3) User Interaction Interception: This attack is possible due to inconsistent click-event reception. A malicious landlord can launch a user interaction interception attack on his cross-origin tenant by inserting display elements below a cross-origin tenant image. In a webpage containing mutually distrusting principals, each principal's actual content as well as the user interaction with the principal's content are private to that principal (due to the Same Origin Policy). Therefore, the browser must not allow unauthorized observation by a principal on a user's interaction with another tenant.

A malicious landlord attacker can intercept user interaction with an opaque cross-origin image ad with a click event in a browser that gives priority to the user events (such as `onclick`, `onmouseover`) of elements situated below the image. The expected behavior of `onclick` on the image is navigation of user's browser to the advertiser's webpage. A user's interaction with the ad on the malicious landlord's page is private to the advertiser because of the Same Origin Policy. To snoop on the user interaction with the tenant, the landlord fills the entire screen area below the image ad with buttons that have an `onclick` event defined. If a user visits the landlord's website from a vulnerable browser and clicks on the image ad, the click event of the buttons below the image will be executed. This browser behavior will allow a malicious landlord to monitor user interaction with the honest tenant.

3.3 Analysis

Android Mobile, Android tablet on Xoom, Nokia Mini-Map and Opera Mini browsers are susceptible to all the attacks; whereas, none of the desktop browsers are susceptible to any of the attacks. We found discrepancies between browsers made by the same vendors. For instance, while Opera Mini is susceptible to all of the attacks discussed in this section, neither the Opera desktop nor Opera Mobile browsers are vulnerable. However, this behavior does not indicate that Opera Mobile enforces all the same policies implemented in Opera desktop as seen in Section 4.

These experiments demonstrate that there are a number of ways in which user actions can be intercepted by hidden and potentially malicious objects when rendered by many popular mobile web browsers. However, as our next set of tests demonstrates, there are more direct ways by which malicious objects can elicit direct user interaction.

4 Boundary Control

Many websites contain one or more cross-origin tenants in the form of ads or widgets. Websites (landlord) rely on the browsers to restrict a tenant's dimensions to the display area as defined by the landlord. However, if a browser allows a malicious ten-

ant to control its own dimensions (display ballooning), the tenant can easily expand its own boundaries, completely disregarding the dimensions specified by the cross-origin landlord. *This lack of boundary control allows the tenant to dominate the constrained mobile screen and intercept a user's intended interaction with the landlord.* We discuss details of the discovered vulnerability and then describe potential attacks.

4.1 Experimental Evaluation

Mobile and tablet browsers: The Android mobile, iPhone and iPad2 Safari, Opera Mini and Opera Mobile browsers allow an iframe to stretch its own dimensions to fit the content inside the iframe. Even if the landlord specifies the dimensions of the iframe, the cross-origin tenant can change them by putting more content in the iframe. By altering the iframe's dimensions, the tenant's iframe does not alter the layout of the original page; rather all other elements on the screen are adjusted around the new dimensions of the iframe while retaining the original relative layout.

Desktop browsers: We observe that desktop browsers restrict the boundaries of a cross-origin tenant to those defined by the landlord. Instead of expanding, these browsers add scroll bars to the contained iframes, allowing the user to scroll the iframes to access the content not immediately visible due to the boundary restrictions. Therefore, the phishing and password stealing attacks are not possible on desktop browsers.

4.2 Attacks

We illustrate two attacks that take advantage of incorrect boundary control.

1) Display Ballooning $\rightarrow$ Phishing: Display ballooning allows a malicious website principal to push legitimate content far outside of the view of the user (an attack made acute by the general lack of visible scroll bars), thereby causing a client to interact with a seemingly benign but actually dangerous function.

Consider the iGoogle mashup webpage (landlord) containing each widget (tenant) inside an iframe. As shown in Figure 3, an honest user innocently adds a malicious widget (ATTACKER) to his profile. ATTACKER is placed "North" of the honest widget Amazon, which shows online deals and helps the user purchase the items of his choice. The intention of the malicious tenant is to navigate an honest user to a website of the tenant's choice. To launch the attack, the malicious tenant alters his dimensions, expands his own iframe and masquerades as the Amazon and YouTube widgets, while pushing the real Amazon and YouTube widgets "South", far outside of the user's view. Unless the user scrolls down very far, he is unable to notice the attack. The user perceives the masqueraded Amazon as the real widget and clicks on the deals of the attacker's choice.

The tenant attacker does not necessarily need to know the presence and layout of specific widgets on the victim's personal profile. The attacker can masquerade as any of the default widgets generally included on the mashup website. Unless the victim is very familiar with the layout of his profile, he will trust the masqueraded widget. Additionally, if the malicious widget is published on a well known mashup website, a not-so-careful user may be willing to click on links he finds interesting irrespective of the credibility of the widget presenting the links to him. The phishing attack can work on any mash-up website with a similar layout.

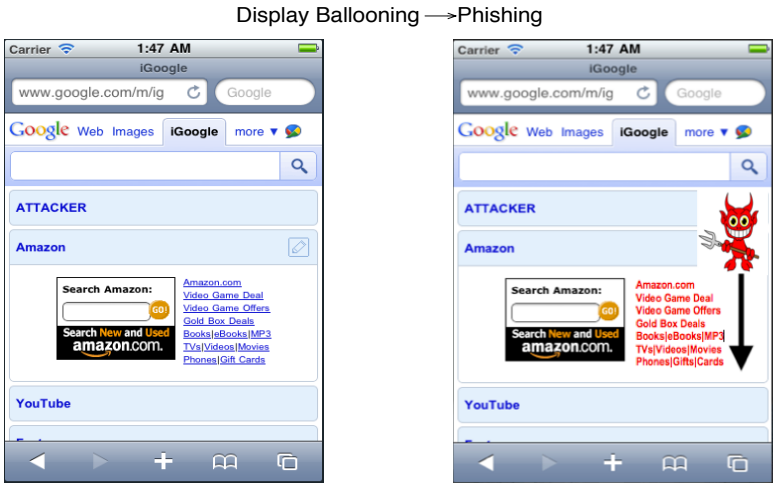

Fig. 3. **Left image**: Layout of the malicious and honest widgets on the mashup webpage. 'ATTACKER' is a malicious widget and Amazon and YouTube are honest widgets; **Right image**: The browser allows a cross-origin tenant to write its own dimensions. The malicious widget expands its own dimensions and masquerades as the honest Amazon and YouTube widgets on the browser. It pushes the honest widgets south and launches a phishing attack on the user. This attack works in the iPhone Safari (pictured), Android mobile, iPad2 Safari, Opera Mini and Opera Mobile browsers.

2) Display ballooning → Password Stealing: Consider a malicious advertisement (tenant attacker) situated to the "North" of the login box of an honest website. The malicious ad can steal a user's credentials by stretching its own dimensions and including a fake login box, which looks exactly the same as the honest website's login box. The real login box would be pushed "South" beyond the bottom of the user's screen. Because the user is not able to see all the content on the screen at the same time, the user will likely enter his credentials in the fake login box.

4.3 Analysis

The Android mobile, iPhone and iPad2 Safari, Opera Mini and Opera Mobile browsers are susceptible to phishing and password stealing as a result of display ballooning. The desktop browsers restrict a tenant iframe's dimensions to those specified by the landlord thereby preventing these attacks.

Browsers made by the same vendor deal with boundary control inconsistently. For example, the Opera Mini, Opera Mobile and iPhone Safari browsers exhibit the same vulnerability, whereas their desktop versions do not. Additionally, while the Android tablet browser on Xoom is susceptible to display ballooning similar to its mobile version, the Android tablet browser on Galaxy behaves like desktop browsers, correctly implementing tenant boundary restrictions.

The experiments in Section 3 and Section 4 demonstrate that none of the desktop browsers are vulnerable to the attacks feasible on mobile browsers. Intuitively, adopting similar policies implemented on desktop browsers will prevent introduction of new vulnerabilities in mobile browsers. However, we show in the next section that reusing desktop browser code without modifications can lead to unexpected vulnerabilities in

mobile browsers, due to adjustments made in mobile browser software for improved usability.

5 Top Level Frame Navigation

The address bar indicates the URL of the viewed webpage and, in some browsers, the current security status. Because of limited screen real-estate, mobile browsers minimize the address bar once a page is rendered, hiding it from the user. This usability concession in mobile browsers directly conflicts with the 'Top-Level Frame Navigation' display policy [21] implemented throughout desktop browsers. This policy governs a principal's ability to navigate principals of other origins. In particular, this policy allows top-level frames (i.e., the landlord) to be navigated by any of its descendants (i.e., tenants) regardless of their origin. Because users can always see the address bar, it is possible for a user to determine if the current destination represents their intended target or a malicious webpage [21]. Accordingly, all desktop browsers allow a user to *always* view the top-level window's address bar.[6] *We show that since mobile browsers do not make the address bar persistently available to a user, browser policies that assume persistent view of address bar for security can be exploited.* We also discuss the differences in our attack and the already studied attacks [30, 37] that exploit non-persistent address bar in mobile browsers, and argue that our attack technique is more dangerous and easier to launch.

5.1 Attack and Experimental Evaluation

A tenant attacker (descendant) can launch a phishing attack if he can navigate the cross-origin top-level window and the top-level window's address bar is not visible to the user.

Consider a webpage `www.honest.com` consisting of a malicious cross-origin advertisement as shown in Figure 4 (left). The `onload` event of the ad is to navigate the top-level window to `www.attacker.com`, which looks exactly the same as `www.honest.com` (Figure 4 (right)) and contains malicious content. When the ad on the honest page is loaded, it navigates the top-level window to the attacker's page. If the user's browser shows the address bar of the top-level window, the user may be able to detect the phishing attack and refrain from interacting with the malicious page. However, if the user's browser does not show the address bar, the user cannot detect the phishing attack.

Experimental Evaluation:
Mobile and tablet browser results: All ten mobile and three tablet browsers allow a tenant principal of any origin to navigate the top-level window to any source.

The iPhone Safari browser minimizes the top-level address bar for better usability once a page is rendered. Moreover, the address bar disappears from view once a user starts interacting with the content on the page. This behavior is seen in all mobile browsers except Blackberry Mango, Chrome Beta, IE Mobile 8 and Nokia Mini-Map. IE Mobile browser persistently displays the address bar only in the portrait mode

[6] The Chrome, Firefox and Safari desktop browsers allow users to hide the address bar through options [33].

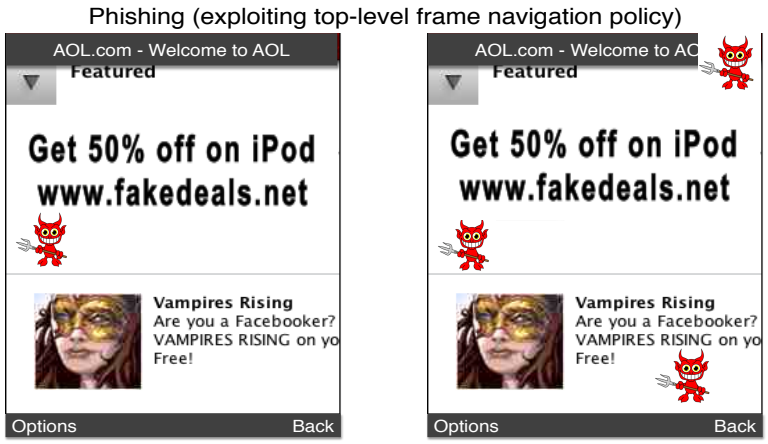

Fig. 4. **Left image**: `www.aol.com` webpage containing a cross-origin malicious advertisement. The browser displays only the 'title' of the page and does not display the address bar.; **Right image**: Due to the top-level frame navigation policy, the malicious ad can redirect the top-level window to `www.attacker.com`, which looks exactly the same as AOL's website, thereby launching a phishing attack. The user cannot detect the attack since the address bar containing the URL of the top window is not included in the mobile browser's view due to space constraint. The Nokia Mini-Map and Blackberry Mango browsers are the most susceptible to this attack. However, all other mobile and tablet browsers (except Chrome Beta and iPad2 Safari) are also susceptible to this attack due to address bar not being persistently available while browsing.

and never in the landscape mode. The Chrome Beta is the only mobile browser allowing persistent view of the address bar. In the Blackberry Mango and Nokia Mini-Map browsers, the address bar of the top-level window is *never* accessible to the user on the screen while browsing. The web address of the top-level window can be viewed from Options → Advanced → Page Info in the Nokia Mini-Map browser. In the Blackberry Mango browser, a user is required to click on the lock icon in the top right corner of the screen to access the address of the webpage. It is difficult for a user to browse to this page info every time he wants to access the top level URL. This makes the Blackberry Mango and Nokia Mini-Map browsers the most susceptible to phishing attacks by navigation of top-level window to malicious pages, since the user can never detect the attack unless he intentionally checks the page information and views the webpage's address.

Interestingly, Safari on the iPad2 differs slightly from its iPhone version in that the address bar is present at all times, enabling users to protect themselves from the phishing attack. However, the Android tablet browsers (both Xoom and Galaxy) exhibit similar behavior as their mobile version and hide the address bar when a user starts interacting with the webpage. Therefore, the Android tablet browsers are susceptible to the phishing attack. We also note that due to the smaller screen size of mobile browsers, the complete URL of a webpage is not necessarily displayed to a user. This makes it even more difficult for a user to make a decision of the credibility of a website at the time of page load, when the address bar temporarily flickers at the top of the browser.

Desktop browser results: All five desktop browsers allow a tenant principal of any origin to navigate the top-level window to any source. However, the desktop browsers

always display the address bar in the window. We note that if Chrome's option to hide the address bar becomes the widespread default, the 'Top-Level Frame Navigation' policy should be reconsidered for all browsers.

5.2 Analysis

When a user interacts with a webpage on a desktop browser by scrolling or zooming, the top-level address bar is always available to the user. However, because of the drastically reduced screen size of mobile devices, removing the address bar from view makes sense in mobile browsers. Because this necessarily pushes the address bar out of the user's sight for most of the time while browsing, the current policy for top-level frame navigation is not appropriate for mobile browsers. We discuss potential solutions to this problem in Section 6.

We note that our phishing attack is significantly different than the existing phishing attacks [30, 37, 40] exploiting address bar hiding in mobile browsers. The existing attacks [30, 37] assume that the user is already on a phishing website, spoof the address bar and then preclude the user from viewing the 'real' address bar using Javascript. Therefore, a successful attack requires an attacker to trick a user into browsing to the phishing website. Our attack does not assume that a user is already on a phishing website. Instead, an attacker can post an advertisement on *any* legitimate website and then redirect the user to a phishing website without requiring any explicit user interaction. This makes our attack more dangerous and feasible as compared to the attacks that require user interaction to launch a phishing website. Any legitimate website hosting cross-origin content becomes vulnerable to our attack. We note that once an attacker redirects a user to a phishing website by exploiting the top-level frame navigation policy, existing address bar spoofing techniques [30,37] can be used to increase the success rate of the attack.

6 Discussion and Potential Solutions

Mobile browsers necessarily make considerations for the constrained platform on which they run. Unfortunately, in the process of porting their software to these devices, vendors have introduced a number of new classes of vulnerabilities. While seemingly unrelated, Table 2 shows that these issues are repeated across many mobile browser vendors. The vulnerabilities presented in this paper are made even more dangerous by the constrained nature of the mobile screen as shown in Section 4.2.

A subset of vendors of the evaluated browsers have confirmed the presence of the vulnerabilities. We note that unavailability of a standard for user event routing and boundary control may be a cause of these vulnerabilities. Identical vulnerabilities were observed in browsers irrespective of the rendering engine used or the manufacturer. For example, the Android Mobile (Webkit) and Opera Mini (Presto) browsers exhibit the same issues; whereas, the five Webkit-based mobile browsers do not demonstrate all of the same vulnerabilities. Intuitively, assuming that browsers built by the same company have some overlap in the development teams suggests that browser components may be reused across platforms. However, the differences in the presence of vulnerabilities in the mobile, tablet and desktop browsers built by the same vendor (e.g., Opera) suggests that new vulnerabilities have been introduced while porting components from existing

Type	Rendering Engine	Browser Name	Attacks		
			Vulnerability - Incorrect handling of user access to overlapping elements	*Vulnerability* - Cross-origin tenant modifying self dimensions	*Vulnerability* - Inconsistent view of address bar
			Click fraud, Login CSRF, User Interaction Interception	Display Ballooning: Phishing, Password Stealing	Phishing
Mobile	Webkit	Android	✓	✓	✓
		Blackberry Webkit			✓
		Chrome Beta			
		iPhone Safari		✓	✓
		Nokia Mini-Map	✓		✓
	Presto	Opera Mini	✓	✓	✓
		Opera Mobile		✓	✓
	Gecko	Firefox Mobile			✓
	Mango	Blackberry Mango			✓
	Trident	Internet Explorer			✓
Tablet	Webkit	Android on Xoom	✓		✓
		Android on Galaxy			✓
		Safari on iPad		✓	
Desktop	Presto, Gecko, Webkit Trident	Opera, Firefox, Safari, Chrome, Internet Explorer			

Table 2. Summary of observed display-related vulnerabilities in candidate browsers and respective attacks possible (A ✓depicts that attack is possible). **1**) Equivalent vulnerabilities exist in mobile and tablet browsers with different rendering engines. **2**) Mobile, tablet and desktop browsers from the same vendor do not necessarily implement the same code to handle display elements in different settings. **3**) Desktop browsers are more compliant with security policies for display.

browser software to a new platform. Whether the discovered vulnerabilities are implementation or design errors in individual browsers is hard to state with certainty. The pervasive nature of the vulnerabilities hints at a more concerning trend.

We propose solutions for the vulnerabilities discussed in this paper. Browsers should always route the click, hover and write user events exclusively to the top element in a stack of overlapped elements. This will provide consistency in handling user event routing and also prevent the attacks discussed in Section 3.2. Secondly, the attacks possible due to erroneous boundary control can be prohibited by restricting dimensions of tenant iframes to those specified by the landlord irrespective of the origins of the tenant and landlord. We note that the evaluated desktop browsers have implemented preventive measures against the attacks discussed in the paper.

Although borrowing desktop browser policies addresses the vulnerabilities in user event routing and boundary control, the small screen size of mobile devices demands more restrictive policies than those implemented in desktop browsers to prevent the phishing attack discussed in Section 5. We propose using Gazelle's top-level frame navigation policy [47] allowing only tenants with the same origin and the user to navigate the top-level window. This approach would better balance issues of usability, specifically screen real-estate, and security. A more extreme solution would be removing support for the top-level frame navigation policy from mobile browsers; however, legitimate webpages relying on this mechanism for functionality may break. Offloading security decisions to the cloud [16] would be another alternative solution to the generic problem of tension between security and usability on small mobile screens. Most critically, borrowing the top-level frame navigation policy to the mobile environment is evidence that security and usability teams are not interacting closely enough with each other. Any solutions should be applied with input from both groups.

The relevance of our observations goes well beyond web browsing. A significant amount of research effort has recently focused on the security of mobile applications [27–29]. These studies have generally centered around applications built for specific platforms. However, an increasing number of applications are becoming highly dependent on the browser. In particular, applications by a number of popular companies (e.g., ESPN, Facebook) are actually wrappers around the browser and point their users to specific webpages within a target domain. The advantage to this approach is that it allows companies to ensure a relatively consistent user experience across all platforms with minimal development effort. As a consequence, however, such "applications" now also potentially become vulnerable to the kinds of attacks discussed in this paper.

7 Related Work

Desktop browsers have been shown to be vulnerable to a variety of attacks in the past including Cross Site Scripting [19], Cross Site Request Forgery [22], clickjacking [3, 4, 39] and phishing. In addition to weak security policies, implementation errors in the browser code [15, 25], inconsistencies in access control policies [41], slow adoption of security techniques [48] and incorrect handling of privileges in browser extensions [20] further increase the threats to the browser and the user. To protect browsers from attacks, a range of defenses have been proposed including implementing new HTTP headers [22], enforcing new security policies [31, 34, 42] and algorithms [14, 19] and development of tools to find potential security vulnerabilities in browsers [18].

Providing strong isolation between cross-domain principals in a browser is another defense technique proposed by researchers in the past. The OP Web browser [32] was the first to design a small browser kernel to enforce new browser security features and handle resources. Gazelle [47] and Chrome [24] also proposed new browser architectures for separating the functionality of the browser from security mechanisms and policies. Tang et al [43] continued the design philosophy through the Illinois browser OS by directly mapping browser abstractions to hardware abstractions.

Web browsers have become one of the most popular applications on today's smart phones. In addition to malicious mobile applications affecting user privacy [27, 28] and potentially harming the cellular network [45], the increasing user base of mobile platforms has made mobile browsers an attractive target for attackers [2,5,6,8,11,30,37,40]. Moreover, mobile browsers implement only a subset of the recommended SSL indicators from the desktop world thus eliminating the opportunity for even expert users to avoid attacks such indicators might signal [17]. Researchers have already begun to think about defending against attacks on mobile phones using smart CDNs [35]. Although mobile browsers will be targets of security attacks in the coming years, security issues in mobile browsers will be new since the devices have serious limitations compared to desktops. However, a large-scale security analysis of the differences between mobile and desktop browsers has not yet been performed.

8 Conclusion

Constrained screen size fundamentally changes the browsing experience on mobile phones. Crowded layout, the inability to consume large amounts of content concurrently and the difficulty in discerning boundaries between different objects on a webpage make

it hard for users to browse the web in the manner to which they are accustomed. In response to these problems and to alleviate these difficulties, mobile browsers have been changed significantly from their desktop counterparts. However, the impact of these changes on security has not been studied. In this paper, we perform the first large-scale comparison of display security between the most popular mobile, tablet and desktop browsers and demonstrate that the differences are far from simply cosmetic. We identify and implement a number of attacks based on two new classes of vulnerabilities found only on mobile and tablet browsers, and then present solutions to address the vulnerabilities. We then identify a third class of vulnerability that exploits the small screen size of mobile devices and a universally implemented policy in all browsers. Our results and feedback from browser vendors exemplify that new vulnerabilities have been introduced while porting browser software to mobile platforms and that usability should be considered while designing solutions instead of blindly porting desktop browser code to the mobile environment.

References

1. 150 Highest Paying Adsense Keywords Revealed! http://earns-adsense.blogspot.com/2008/04/150-highest-paying-adsense-keywords.html
2. Android Browser Exploit. http://threatpost.com/en_us/blogs/researcher-publishes-android-browser-exploit-110810
3. Chrome, Firefox get clickjacked. http://www.zdnet.com.au/chrome-firefox-get-clickjacked-339294633.htm/
4. Facebook clickjacking. http://personalmoneystore.com/moneyblog/2010/08/18/facebook-clickjacking-social-network-scams/
5. iPhone overflow clickjacking. http://ejohn.org/blog/clickjacking-iphone-attack/
6. iPhones Safari Vulnerable To DoS Attacks. http://www.iphonebuzz.com/iphone-safari-dos-bug-discovered-162212.php
7. Mobile Browser Market Share. http://gs.statcounter.com/#mobile_browser-ww-daily-20120307-20120405
8. Overflow clickjacking. http://research.zscaler.com/2008/11/clickjacking-iphone-style.html
9. Paying by the Click. http://www.nytimes.com/2007/10/15/us/15bar.html?ref=us
10. Same-origin policy. http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy
11. Web-based Android attack. http://www.infoworld.com/d/security-central/security-researcher-releases-web-based-android-attack-317?source=rss_security_central/
12. Opera Presto 2.1 - Web standards supported by Opera's core. http://dev.opera.com/articles/view/presto-2-1-web-standards-supported-by/ (2011)
13. The WebKit Open Source Project. http://webkit.org/ (2011)
14. Adida, B.: Beamauth: two-factor web authentication with a bookmark. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2007)
15. Aggarwal, G., Bursztein, E., Jackson, C., Boneh, D.: An Analysis of Private Browsing Modes in Modern Browsers. In: USENIX Security Symposium (2010)
16. Amrutkar, C., van Oorschot, P.C., Traynor, P.: An Empirical Evaluation of Security Indicators in Mobile Web Browsers. Georgia Tech Technical Report GT-CS-11-10 (2011)
17. Amrutkar, C., van Oorschot, P.C., Traynor, P.: Measuring SSL Indicators on Mobile Browsers: Extended Life, or End of the Road? In: Proceedings of the Information Security Conference (ISC) (2012)
18. Bandhakavi, S., King, S.T., Madhusudan, P., Winslett, M.: VEX: Vetting Browser Extensions For Security Vulnerabilities. In: Proceedings of the USENIX Security Symposium (SECURITY) (2010)
19. Barth, A., Caballero, J., Song, D.: Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland (2009)
20. Barth, A., Felt, A.P., Saxena, P., Boodman, A.: Protecting Browsers from Extension Vulnerabilities. In: Proceedings of the 17th Network and Distributed System Security Symposium (NDSS) (2010)
21. Barth, A., Jackson, C.: Protecting Browsers from Frame Hijacking Attacks. http://seclab.stanford.edu/websec/frames/navigation/
22. Barth, A., Jackson, C., Mitchell, J.C.: Robust Defenses for Cross-Site Request Forgery. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2008)
23. Barth, A., Jackson, C., Mitchell, J.C.: Securing frame communication in browsers. In: Proceedings of the USENIX Security Symposium (SECURITY) (2008)
24. Barth, A., Jackson, C., Reis, C., The Google Chrome Team: The security architecture of the chromium browser. http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf

25. Barth, A., Weinberger, J., Song, D.: Cross-origin javascript capability leaks: detection, exploitation, and defense. In: Proceedings of the USENIX Security Symposium (SECURITY) (2009)
26. Blog, G.M.A.: Smartphone user study shows mobile movement under way. http://googlemobileads.blogspot.com/2011/04/smartphone-user-study-shows-mobile.html (2011)
27. Egele, M., Kruegel, C., Kirda, E., Vigna, G.: PiOS: Detecting Privacy Leaks in iOS Applications. In: Proceedings of the ISOC Networking & Distributed Systems Security (NDSS) Symposium (2011)
28. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI) (2010)
29. Enck, W., Octeau, D., McDaniel, P., Chaudhuri, S.: A Study of Android Application Security. In: Proceedings of the USENIX Security Symposium (2011)
30. Felt, A.P., Wagner, D.: Phishing on Mobile Devices. In: Proceedings of the IEEE Web 2.0 Security and Privacy Workshop (W2SP) (2011)
31. Grier, C., King, S.T., Wallach, D.S.: How I Learned to Stop Worrying and Love Plugins. In: In Web 2.0 Security and Privacy (2009)
32. Grier, C., Tang, S., King, S.T.: Secure Web Browsing with the OP Web Browser. In: Proceedings of the IEEE Symposium on Security and Privacy (Oakland) (2008)
33. Gus Andrews: Has the address bar had its day? http://www.netmagazine.com/features/has-address-bar-had-its-day
34. Huang, L.S., Weinberg, Z., Evans, C., Jackson, C.: Protecting browsers from cross-origin CSS attacks. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2010)
35. Livshits, B., Molnar, D.: Empowering Browser Security for Mobile Devices Using Smart CDNs. In: Proceedings of the Workshop on Web 2.0 Security and Privacy (W2SP) (2010)
36. Luttrell, M.: Majority of users prefer mobile browser over apps. http://www.tgdaily.com/mobility-brief/55884-majority-of-users-prefer-mobile-browser-over-apps (2011)
37. Niu, Y., Hsu, F., Chen, H.: iPhish: Phishing Vulnerabilities on Consumer Electronics. In: Usability, Psychology, and Security (2008)
38. Ruderman, J.: Same Origin Policy for JavaScript. http://www.mozilla.org/projects/security/components/same-origin.html
39. Rydstedt, G., Bursztein, E., Boneh, D., Jackson, C.: Busting Frame Busting: A Study of Clickjacking Vulnerabilities at Popular Sites. In: Proceedings of the IEEE Web 2.0 Security and Privacy Workshop (W2SP) (2010)
40. Rydstedt, G., Gourdin, B., Bursztein, E., Boneh, D.: Framing Attacks on Smart Phones and Dumb Routers: Tap-jacking and Geo-localization Attacks. In: Proceedings of the USENIX Workshop on Offensive Technology (WOOT) (2010)
41. Singh, K., Moshchuk, A., Wang, H.J., Lee, W.: On the Incoherencies in Web Browser Access Control Policies. IEEE Symposium on Security and Privacy (Oakland) (2010)
42. Tang, S., Grier, C., Aciicmez, O., King, S.T.: Alhambra: a system for creating, enforcing, and testing browser security policies. In: Proceedings of the International Conference on World Wide Web (www) (2010)
43. Tang, S., Mai, H., King, S.T.: Trust and protection in the Illinois browser operating system. In: Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI) (2010)
44. The Open Mobile Alliance: Wireless Application Protocol (WAP) 1.0 Specification Suite. http://www.wapforum.org/what/technical_1_0.htm (1998)
45. Traynor, P., Lin, M., Ongtang, M., Rao, V., Jaeger, T., La Porta, T., McDaniel, P.: On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2009)
46. Wang, H.J., Fan, X., Howell, J., Jackson, C.: Protection and communication abstractions for web browsers in MashupOS. In: Proceedings of 21st ACM SIGOPS symposium on Operating systems principles (2007)
47. Wang, H.J., Grier, C., Moshchuk, A., King, S.T., Choudary, P., Venter, H.: The Multi-Principal OS Construction of the Gazelle Web Browser. In: Proceedings of the USENIX Security Symposium (SECURITY) (2009)
48. Zhou, Y., Evans, D.: Why Aren't HTTP-only Cookies More Widely Deployed? In: Proceedings of the IEEE Web 2.0 Security and Privacy Workshop (W2SP) (2010)